Lab wk 2-2: Finding Connected Components and Testing if Bicolorable.

Finding connected components and determining whether a graph is bipartite (also known as Bicolorable) are important problems in computer science. A **coloring** is an assignment of a color to each vertex in the graph so that no two adjacent vertices have the same color.

Implement an algorithm based on Breath First Search (Levitin page 126) to determine the connected components of a graph and whether it is bipartite. That is, can the vertices of the graph be assigned labels, say red and black, such that no two adjacent vertices have the same label. (Thus the vertex set can be divided into two disjoint subsets where all the edges go from a vertex in one subset to a vertex in the other subset.)

You may assume that the graph is **undirected and does not contain self-loops (**edges from a vertex to itself). You **must** use the graph class, **GraphStart3**, supplied as a starting point.

Your goal is to add two methods to this class. Your new class should be called **MyGraph** it should contain two methods **connectCheck,** that includes checks for connectivity and the computes the number of connected components, and a method **bipartiteCheck** that checks to see if the graph is bipartite.

- public ArrayList<HashSet<Integer>> **connectCheck()** returns an ArrayList of HashSet where the first set contains the number of connected components, followed by sets containing the vertices of the connected components. The connected components should be the ordered in which they are discovered by your algorithm.
- public boolean **bipartiteCheck()** returns true if the graph is bipartite and false if it is not bipartite.

An input file consists of one test case. A readfile_graph method is provided

- Each test case begins with a line containing 1 or 0 to tell your program if the graph is directed or undirected. **For this assignment the line will always contain 0.**
- The second line contains an integer **n**, that is the number of vertices in the graph to be tested, where 1<n<200. Each vertex is represented by a number from 1 to n
- The third line is the number of edges, **e**
- This is followed by **e** lines each containing a pair of integers each integer between 1 and n that represents an edge between the vertices represented by the numbers.

Example **Input:**

```
0
9
8
1 2
1 3
1 4
1 5
6 7
7 9
9 8
8 6
```

If this file is used to create a MyGraph g

A call to g.bipartiteCheck() should return

```
true
```

A call to g.connectCheck() should return

```
[{2} , {1 2 3 4 5}, {6 7 8 9}]
```

Be sure to test you program thoroughly. Make additional test cases and modify the print graph routine to print out the coloring of the graph to ease in debugging. **Suggestion:** Make up very simple graphs that test possible variation of the results.